Letting the Data Lead the Code

Darryl Bryk
U.S. Army RDECOM-TARDEC
Warren, MI 48397

Introduction

Sometimes the data format used can radically simplify the code. A simple change to the way data is stored in a file can make a big difference in the code required to read it. This was made apparent recently with some Visual Basic code which would locate a list of numbers based on certain search criteria. The code was about one thousand lines and needed to be converted to C# for a new application. It was time to rethink how this could be simplified.

The original data format was just a series of numbers stored in an Excel spreadsheet accessed by variable names. Through a long series of if-then-else and case statements the code determined which data to read from the file. The data-search hierarchy was in the source code. It was found that by changing the data format in a hierarchically structured way the code could be drastically simplified. After changing the data format the search function was simplified to about forty lines of code. Although this article will describe a specific data type search algorithm, it may inspire the developer to think more about the data-code design.

Background

The original code operated by searching for specific sensor information from which it could determine which data should be read from the spreadsheet. There are three different sets of data (e.g. Set1, Set2, Set3) from which to search, and each set has data for three types of machines (M1, M2, M3). Each machine has from fifty to sixty sensors which could be in seven different positions (Pos1, … Pos7), some of which could have upper or lower positions (UP, LO), and each sensor could be of three different types (F, M, A), and have two to three axes (x, y, z). Also, some data pertains to both +/- axes and some to just the + axis or – axis. As an example, suppose a search request was for the +z-axis, type F sensor, position Pos1UP, for the M1 machine, and using the Set1 criteria. The original code would call the Set1 handling function and go through the if-then-else statements to locate Pos1UP, the F +z-axis, and then the M1 machine, and then the criteria number could be read. Once the search was satisfied the data lookup was very simple. The complicated decision-search algorithm was all in the code. An example section of the VB code is

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE	2. REPORT TYPE	3. DATES COVERED
31 AUG 2015		**00-00-2015 to 00-00-2015**

4. TITLE AND SUBTITLE	5a. CONTRACT NUMBER
Letting the Data Lead the Code	5b. GRANT NUMBER
	5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)	5d. PROJECT NUMBER
DARRYL BRYK	5e. TASK NUMBER
	5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)	8. PERFORMING ORGANIZATION REPORT NUMBER
US Army RDECOM-TARDEC,6501 E. 11 Mile Road,Warren,MI,48397-5000	
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)	10. SPONSOR/MONITOR'S ACRONYM(S)
	11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT
See Report

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT	b. ABSTRACT	c. THIS PAGE			
unclassified	**unclassified**	**unclassified**	**Same as Report (SAR)**	**4**	

shown below where `bInStr ()` returns a Boolean indicating whether the passed string is found in the string, `IV` was a list that would accumulate the pertinent data values, and `_IA_M1_Pos1` is an example variable name in the Excel data file where the data resided. The `Sht.[_IA_M1_Pos1].Item(1, 1)` and `Sht.[_IA_M1_Pos1].Item(1, 2)` retrieve the values at `_IA_M1_Pos1` and one cell to the right, respectively.

```
With Sht
    If (bInStr(str, "Pos1")) Then
        Select Case (sMachine)
            Case "M1"
                With .[_IA_M1_Pos1]
                    Call IV.Add("Set1 " + sMachine, .Item(1, 1) & " " & .Item(1, 2), "H")
                End With
            Case "M2"
                With .[_IA_M1_Pos1]
                    Call IV.Add("Set1 " + sMachine, .Item(1, 1) & " " & .Item(1, 2), "H")
                End With
            Case "M3"
                With .[_IA_M1_Pos1]
                    Call IV.Add("Set1 " + sMachine, .Item(1, 1) & " " & .Item(1, 2), "H")
                End With
            End Select
    End If
    ElseIf (bInStr(str, "Pos2") And bInStr(str, "UP")) Then
    :
    :
```

Using the Code

By offsetting some of the data complexity to the data file instead of the code, and rewriting the data file in a hierarchically ordered format like this:

<set> <machine> <position> <sensor-axis> (data)

where the number in parenthesis is the sought after data value(s), the code was drastically simplified to a while-loop with a short series of nested “if-then” statements. The data file is just a text file and each line keeps the same formatting as above. Each “set” type is kept together in the file as contiguous lines so if the search turns up empty by the time a different set is read, the search can stop. When the set is matched, the search loop proceeds deeper.

An example of the C# code is shown below.

```
public static List<int> SFData(string sSet, string sMachine, string sPosition,
                               string sUPLO, string sAxis) {
    if (SDFileName == "") return null; // Data file name (class variable)

    List<int> ilist = new List<int>();
    string sopt = sPosition + sUPLO; // To catch UP/LO
    bool bFound = false; // Flag to break loop

    try {
        using (StreamReader reader = new StreamReader(SDFileName)) {
            while (!reader.EndOfStream) {
                string str = reader.ReadLine();
```

```
                if (str != "" && str[0] != '/' && str[0] != '<') { // Skip directives
                    string[] s = str.Split(new char[] { ' ', '(', ',', ')' });
                    if (s[0] == sSet) {
                        if (s[1] == sMachine) {
                            if (s[2] == sPosition || s[2] == sopt) {
                                bFound = true;
                                if (s[3].Contains(sAxis)) {
                                    iv.Add(new List <int>());
                                    int inum;
                                    for (int i = 4; i < s.Length; i++) { // Get numbers
                                        if (s[i] == "//") break; // Comment skip
                                        if (int.TryParse(s[i], out inum))
                                            ilist.Add(inum);
                                    }
                                }
                            }
                        }
                        else if (bFound) break; // Stop reading
                    }
                }
            }
        }
        catch (Exception err) {
            MessageBox.Show("Error - " + err.Message, "SFData()",
                            MessageBoxButtons.OK, MessageBoxIcon.Error);
            return null;
        }

        if (ilist.Count == 0)
            return null;
        else
            return ilist;
    }
```

`SFData` reads the text data file `SDFileName` line by line, first looking for a match for `sSet`, then `sMachine`, then `sPosition` (or `sopt` which includes position information "UP" or "LO"), and finally `sAxis`. The read-in string is parsed with `Split()` for white space, commas, and parentheses, and comments and directives in the file are flagged by being preceded with "//" and "<", respectively, so these lines can be skipped. A for loop reads in the data until the end of the string, since data can be from one to four numbers. The function returns a list of the appropriate data numbers, or null, if the search was unsuccessful.

Points of Interest

The code was drastically simplified, is much easier to maintain, and is generic enough to work for all the different types of sets, sensors, etc. The data file maintenance is also simplified and can be updated without making changes to the code as long as the basic order and format stay the same.

A side advantage of re-configuring the data file allowed for directives (enclosed in "<>") which are used by the application to load menu options for selecting the data set. The application can then add menu options at run-time based on the data file that is issued to the customer, and data sets can be tailored to a particular customer. In addition, the data file itself is fairly straight forward to update. In fact, the customer could edit the data file to add their own data criteria and menus.

Conclusion

The original code of decision making if-then-else blocks may have run faster, since the new code has to read through lines of strings until the criteria is found or reaches the end of file, but since the data file is less than 400 lines, and the search stops when the search is satisfied, the speed difference is probably very minimal. The move to the new code most dramatically simplified the code maintenance however, and allows for much simpler changes and additions to the data file, which can now even be done without the need of updating code and re-compiling the application. Although this article described a specific type of data search, it may inspire the developer to re-look at ways to simplify their own data-code interface.